L2 – Instruction Set Architecture

EECS 370 – Introduction to Computer Organization – Winter 2022

Announcements

- Homework 1
  - Released: January 11th
  - Due: Thursday January 24st at 11:55pm
- Project 1
  - Released: January 11th
  - Due: Thursday January 27th at 11:55pm
- Midterm Exam date and time
  - March 10th at 7:00pm– 9:00pm (Rooms TBD)
  - Declare all exam conflicts and SSD requests by January 31st

Recap from last lecture

- Welcome to 370!
- Course Logistics
- Course Overview

Today's Agenda

- Instruction Set Architecture Introduction
- Instruction Encoding
- Instruction Decoding

- Readings
  - Patterson and Hennessy – ARM Edition
  - Sections 2.2, 2.3, 2.4, 2.5

L2_1 – Instruction Set Architecture - Introduction

EECS 370 – Introduction to Computer Organization – Winter 2022

Learning Objectives

- To identify the information of an Instruction Set Architecture (ISA)
- Be able to identify trade-offs relevant to ISA design
- Identify basic, course-granularity operation of a computer
  - Fetch, Decode, Execute

Instruction Set Architecture (a.k.a. Architecture)

Instruction Set Architecture (ISA)

- An abstract interface between the hardware and the lowest-level software that encompasses all the information necessary to write a machine language program that will run correctly, including instructions, registers, memory accesses, I/O, and so on.

Instruction Set Architecture (ISA)
Includes anything programmers need to know to make a binary program work correctly

Instruction Set Architecture (ISA)
Defines interface between hardware and software

1

ISAs

Application software

Compilers

Architecture – a.k.a. ISA (Binary Instruction Set)
- Platform-specific (x86, Arm, risk 5)
- A limited set of assembly language commands available by hardware
  - e.g., `ADD`, `LOAD`, `STORE`, `RET`

The software / hardware divide

Microarchitecture – hardware implementation of ISA (implementation of that Binary Instruction Set.)
- Intel Core i9/i7/i5 implements x86 ISA (desktop/laptop)
- Apple A9 implements ARM v8-A ISA (iPhone)

Circuits

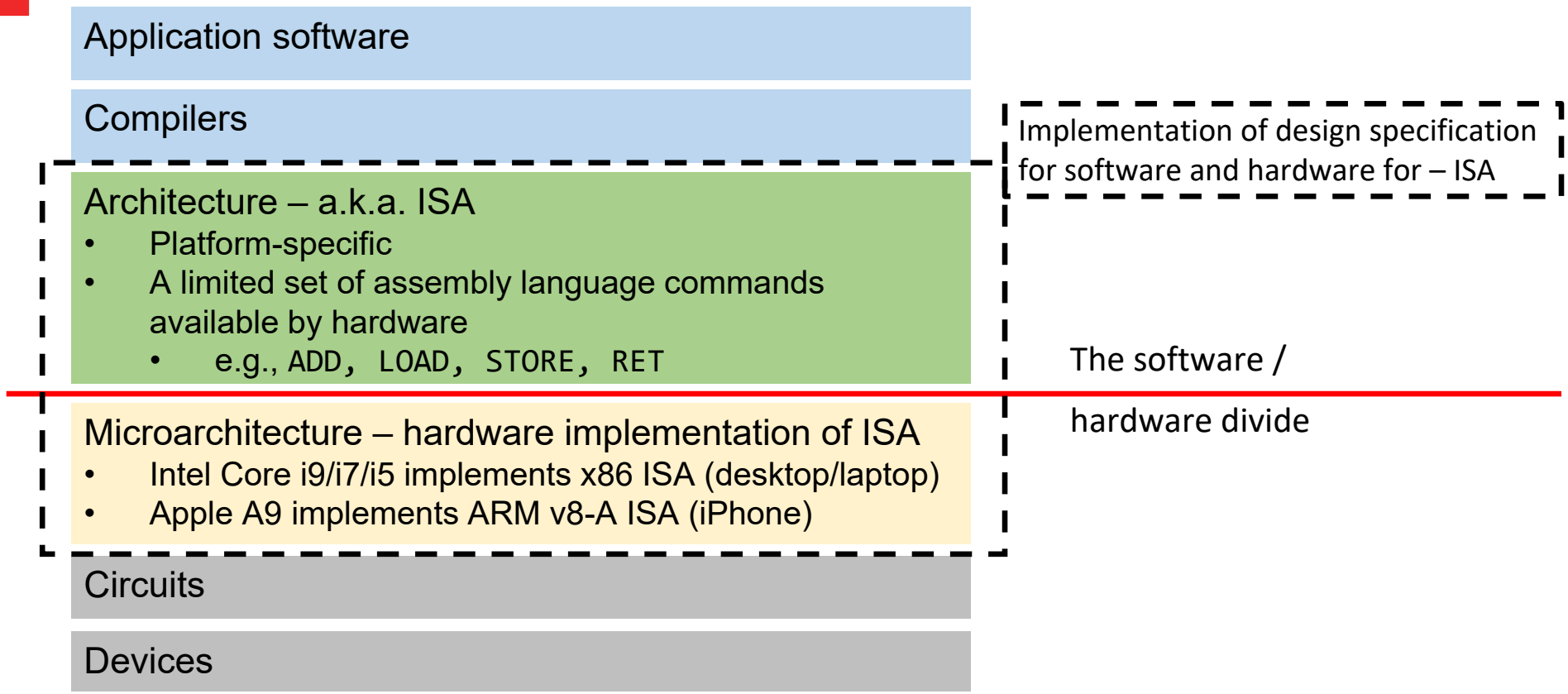
Devices

Slide 8

1 William Arthur, 8/31/2020

2

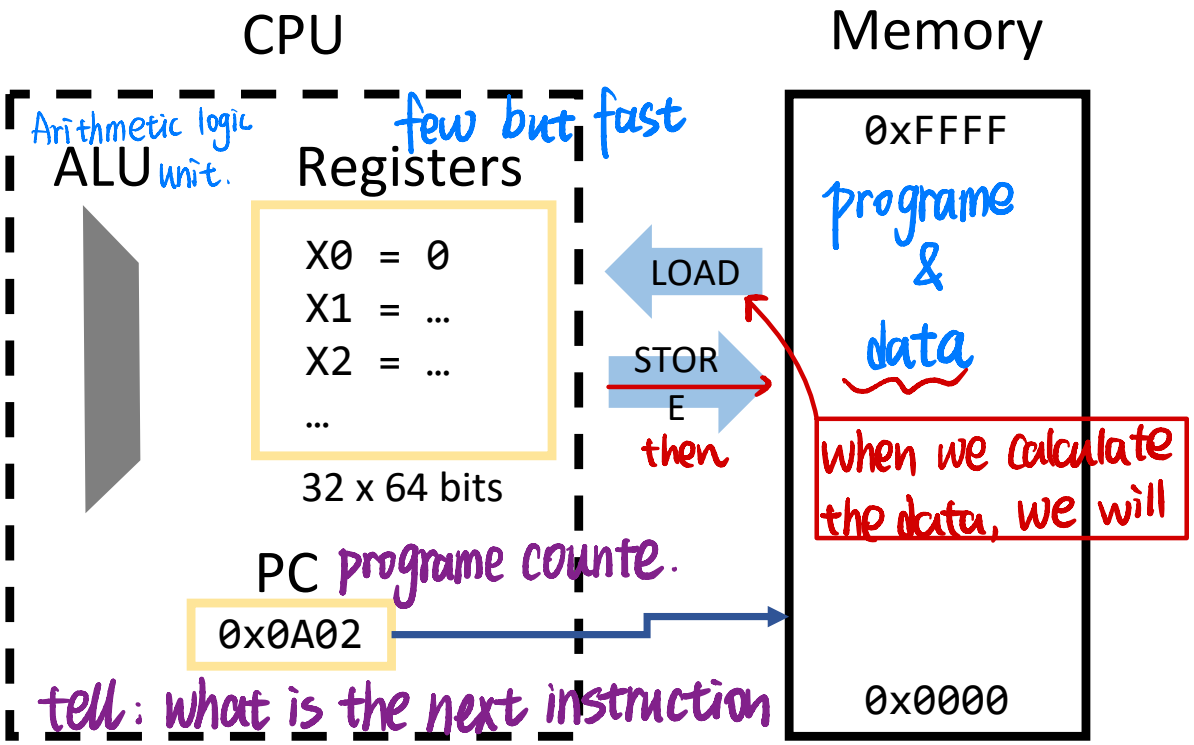
ISAs


Application software
Compilers
Architecture – a.k.a. ISA
• Platform-specific
• A limited set of assembly language commands available by hardware
• e.g., ADD, LOAD, STORE, RET
Microarchitecture – hardware implementation of ISA
• Intel Core i9/i7/i5 implements x86 ISA (desktop/laptop)
• Apple A9 implements ARM v8-A ISA (iPhone)
Circuits
Devices
Implementation of design specification for software and hardware for – ISA
The software / hardware divide

Slide 9

2 William Arthur, 8/31/2020

(Simplified) System Organization



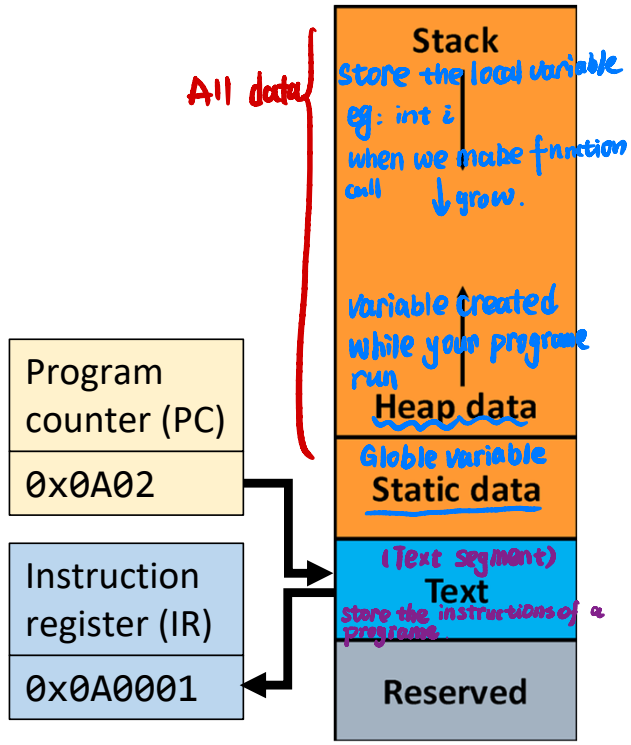
CPU – Central Processing Unit

ALU – Arithmetic Logic Unit, executes instructions

PC – Program Counter, holds address (in memory) of next instruction

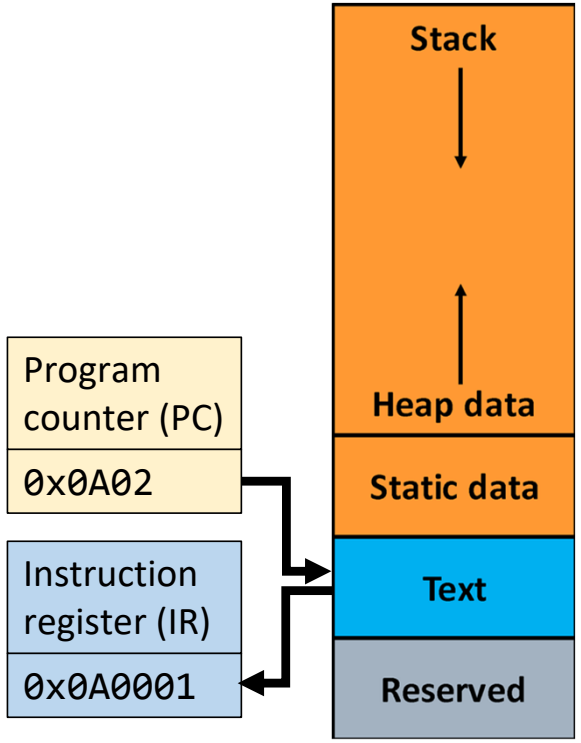
von Neumann Architecture

- von Neumann Architecture
  - Data and instructions are stored in the same memory
  - Programs (instructions) can be viewed as data – simplifies storage
  - Data can be viewed as instructions – complicates security



von Neumann Architecture

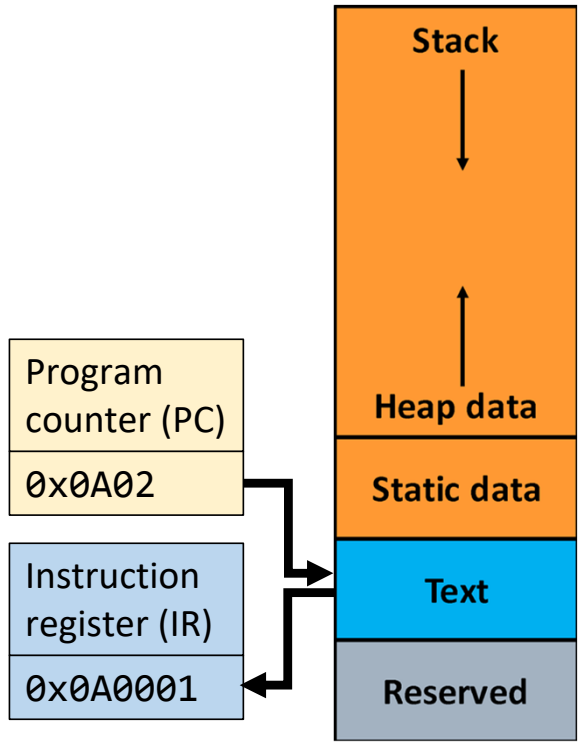
- von Neumann Architecture
  - Data and instructions are stored in the same memory
  - Programs (instructions) can be viewed as data – simplifies storage
  - Data can be viewed as instructions – complicates security
- Instructions are stored sequentially in memory
  - Accessed by the program counter (PC) —it contains the address/location of the instruction the hardware is executing
  - The PC is simply incremented to "point to" the next instruction
  - "jumps" / "branches" override fetching the sequential next instruction
  - Terminology: Jumps are usually unconditional, and branches are conditional on a flag being checked
    - there are conditional jumps….



for loop: ① Check the condition, if true, then run the body
↳ conditional branch.
② from the bottom of the for loop directly back to the begin.
↳ unconditional jump.

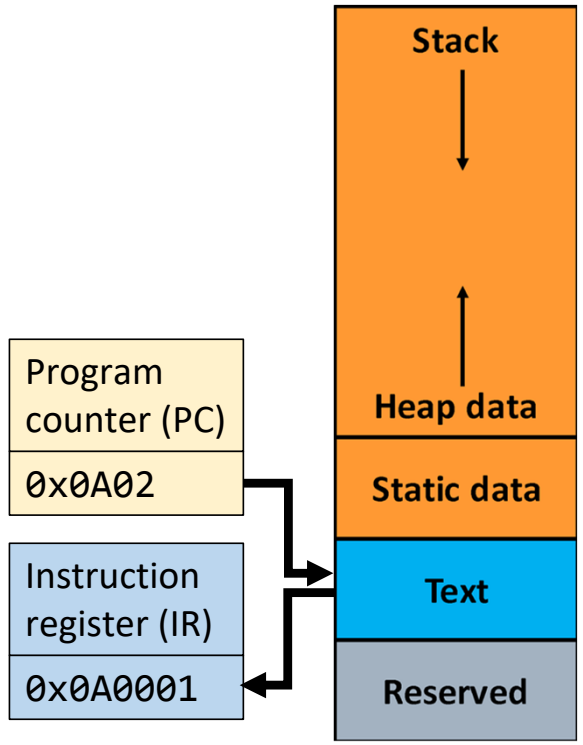
von Neumann Architecture

1. Fetch – get the next instruction. Use the PC to find instruction, put into instruction register (IR).
    1. The PC is changed to “point” to the next instruction in the program
    2. Assume that the next instruction is sequential and contiguous in memory



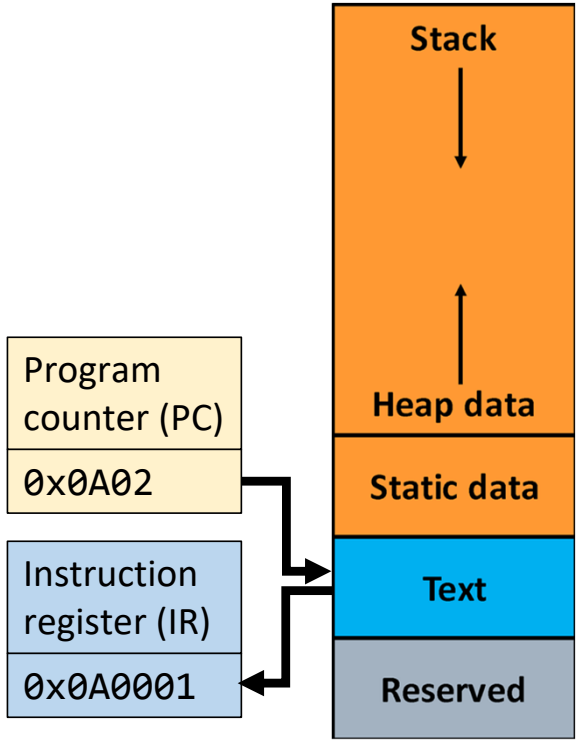
von Neumann Architecture

1. Fetch – get the next instruction. Use the PC to find instruction, put into instruction register (IR).
    1. The PC is changed to “point” to the next instruction in the program
    2. Assume that the next instruction is sequential and contiguous in memory
2. Decode – control logic examines the contents of the IR to decide what functionality to perform



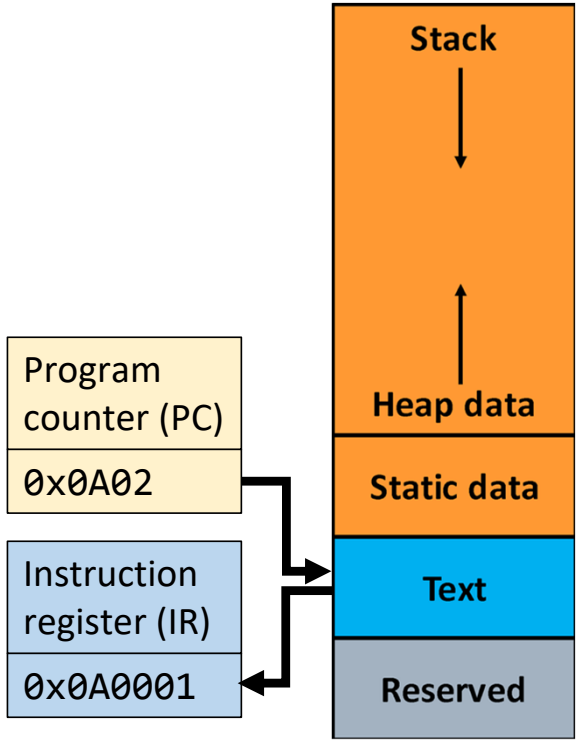
von Neumann Architecture

1. Fetch – get the next instruction. Use the PC to find instruction, put into instruction register (IR).
    1. The PC is changed to “point” to the next instruction in the program
    2. Assume that the next instruction is sequential and contiguous in memory
2. Decode – control logic examines the contents of the IR to decide what functionality to perform
3. Execute – the outcome of the decoding process dictates:
    1. An arithmetic or logic operation on data
    2. The kind of access to data in the same memory as instructions
    3. OR the outcome is a change of contents of the PC



von Neumann Architecture

1. Fetch – get the next instruction. Use the PC to find instruction, put into instruction register (IR).
    1. The PC is changed to “point” to the next instruction in the program
    2. Assume that the next instruction is sequential and contiguous in memory
2. Decode – control logic examines the contents of the IR to decide what functionality to perform
3. Execute – the outcome of the decoding process dictates:
    1. An arithmetic or logic operation on data
    2. The kind of access to data in the same memory as instructions
    3. OR the outcome is a change of contents of the PC
4. Go to step 1



Instruction Set Architecture – Design Space 1

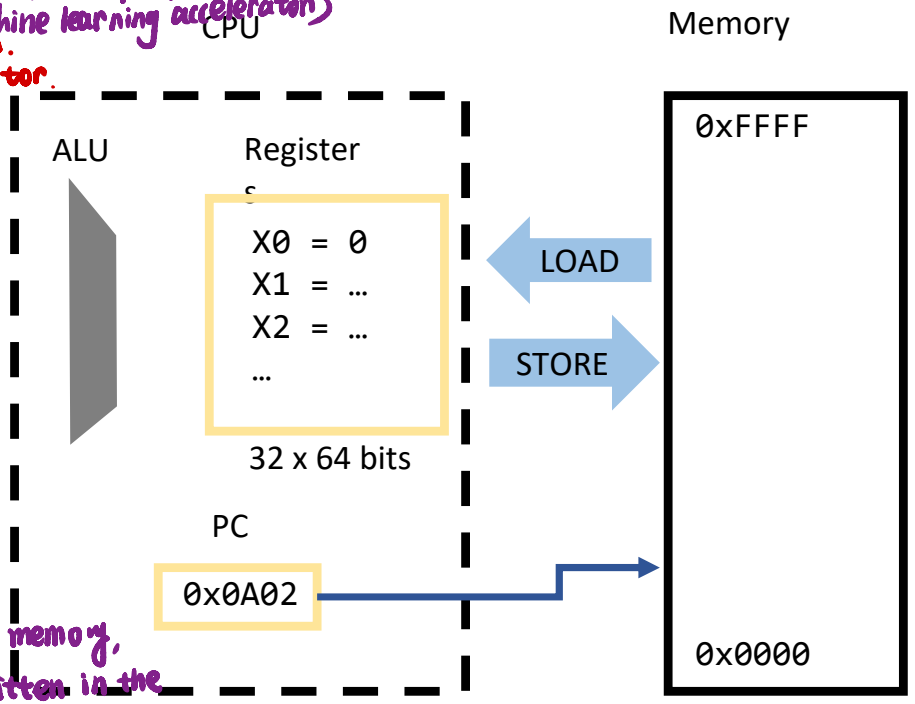
In order to fast the computer, we need to build new accelerators, and they have their own instruction set

for specific accelerator, you build specific instruction.

a cpu for a specific domain (machine learning accelerator)

eg: sigma for machine learning accelerator.

- What instructions should be included?
  - add, multiply, divide, sqrt [functions]
  - branch [flow control]
  - load/store [storage management]
- What storage locations?
  - How many registers? 32
  - How much memory? 2^{64} address
  - Any other "architected" storage?
- How should instructions be formatted?
  - 0, 1, 2 or more operands?
  - Immediate operands



the value doesn't come from register or memory, it's written in the instruction. (small value)

eg: jvm : java virtual machine : 0 operands.

Do add on the first two thing on the stack, And push the result on the stack.

more general : since every thing we do in math either take one or two operate.

Negative → 1

Devide / subtracte → 2.

38 : 3b

Instruction Set Architecture – Design Space 2

- How to encode instructions?
  - **RISC** (Reduced Instruction Set Computer):
    all instructions are same length (e.g. ARM, LC2K)
    smaller set of simpler instructions
  - **CISC** (Complex Instruction Set Computer):
    instructions can vary in size (Digital Equipment's VAX, x86)
    large set of simple and complex instructions
    they actually do RISC operations
- What instructions can access memory?
  - For ARM and LC2K, only loads and stores can access memory (called a "**load-store architecture**") can only do operations on register.
  - Intel x86 is a "**register-memory architecture**", that is, ~~other~~ any instructions beyond load/store can access memory (flexible)
  - Also Compute in Memory (currently a research topic) – simple operations performed in memory without data moving to/from the processor.

Many Choices, Many ISAs

- Why are there many ISAs?
  - Many problem domains, design constraints (e.g., power), differences of opinion
- How often are new architectures created?
  - New embedded processors are created all the time
  - Existing ISAs are extended for new problem domains
    - X86: MMX, MMX2, SSE, AVX, x87, x64
- Can you design one?
  - Yes!

High-Level to Low-Level Language to Hardware

```
int main() {

  int a, *b,
c;
  c = a + *b;

}
```

C program

Compile

```
.text
.global_main
X1 = LOAD(X2);
X3 = ADD(X1,
X4);
```

Assembly Code

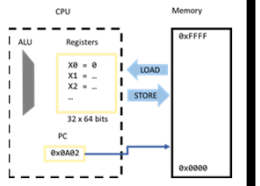
(human readable)

Assemble

```
0x7
456
0xA
16B
```

Machine Code

Load



Machine

L2_2 Assembly and Instruction Encoding

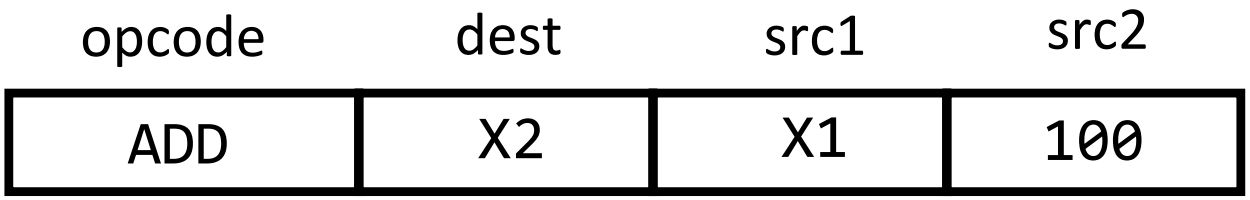
EECS 370 – Introduction to Computer Organization – Winter 2022

Learning Objectives

- To understand the process of encoding an assembly instruction
  - Converting from assembly to machine code
- After completing this video and associated worksheet:
  - You should be able to encode assembly instructions, necessary for Project 1

Assembly Code – Instruction Encoding

- Fields
  - Opcode – What instruction to perform
  - Source (input) operand specifier(s)
    - What data to perform operation on
  - Destination (output) operand specifiers(s)
    - What data to updated



number ⟷ operation.

Execution: value in register X2 = contents reg. X1 + constant 100

Assembly Code - Properties

text representation ⟷ binary representation

- Generally 1-1 correspondence with machine language
- Mnemonic codes facilitate programming
- Labels (symbolic names) names for lines of code (use to jump to those lines of code)
- Direct control of the what processor does
- May execute fast, if you're good at it, but compilers can typically generate better code
- Still hard to use and not portable to other brands of machines

Assembly – ARM Execution Example

Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI (immediate)	X3,	X3,	#3 (consider as constant)	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

Assembly – ARM Execution Example

Given the initial conditions, determine the values of the registers (X1, X2, X3) after executing lines each line of code [1,2,3]. Lines of code are executed sequentially

Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI	X3,	X3,	#3	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

usually use 0~8. store them in instruction is better than load them from another register.

Initial State:	X1 = 25	X2 = -4	X3 = 57

For a number X, 20% we do X+(X+1).

Program

reduce register pressure.

Line	CMD	X1	X2	X3
[1]	ADD X3, X1, X2	25	-4	21
[2]	ADDI X3, X3, #3	25	-4	24
[3]	SUB X2, X3, X1	25	-1	24

Define the first Source operand is a register index, the second Source operand is immediate store in the instruction

Assembly – ARM Execution Example

Given the initial conditions, determine the values of the registers (X1, X2, X3) after executing lines each line of code [1,2,3]. Lines of code are executed sequentially

Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI	X3,	X3,	#3	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

Initial State:	X1 = 25	X2 = -4	X3 = 57

Program

Line	CMD	X1	X2	X3
[1]	ADD X3, X1, X2	25	-4	21
[2]	ADDI X3, X3, #3			
[3]	SUB X2, X3, X1			

Assembly – ARM Execution Example

Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI	X3,	X3,	#3	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

Given the initial conditions, determine the values of the registers (X1, X2, X3) after executing lines each line of code [1,2,3]. Lines of code are executed sequentially

Initial State:	X1 = 25	X2 = -4	X3 = 57

Program

Line	CMD	X1	X2	X3
[1]	ADD X3, X1, X2	25	-4	21
[2]	ADDI X3, X3, #3	25	-4	24
[3]	SUB X2, X3, X1			

Assembly – ARM Execution Example

Given the initial conditions, determine the values of the registers (X1, X2, X3) after executing lines each line of code [1,2,3]. Lines of code are executed sequentially

Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI	X3,	X3,	#3	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

Initial State:	X1 = 25	X2 = -4	X3 = 57

Program

Line	CMD	X1	X2	X3
[1]	ADD X3, X1, X2	25	-4	21
[2]	ADDI X3, X3, #3	25	-4	24
[3]	SUB X2, X3, X1	25	-1	24

Assembly – ARM Execution Example

Program

Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI	X3,	X3,	#3	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

Register	Initial	ADD X3, X1, X2	ADDI X3, X3, #3	SUB X2, X3, X1
X1	25			
X2	-4			
X3	57			

Assembly – ARM Execution Example

Program

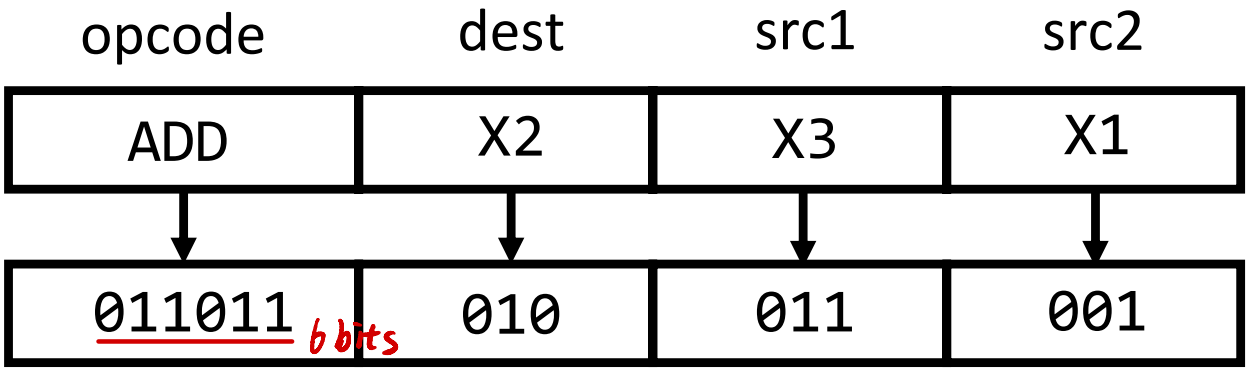
Opcode	Destination Register	Source Reg. 1	Source Reg. 2 / Immediate	Pseudocode
ADD	X3,	X1,	X2	X3 = X1 + X2
ADDI	X3,	X3,	#3	X3 = X3 + 3
SUB	X2,	X3,	X1	X2 = X3 – X1

Register	Initial	ADD X3, X1, X2	ADDI X3, X3, #3	SUB X2, X3, X1
X1	25	25	25	25
X2	-4	-4	-4	-1
X3	57	21	24	24

Assembly – Instruction Encoding

Example ISA (Simplified)

- Instructions are stored as data in memory
- Each instruction is encoded as a number



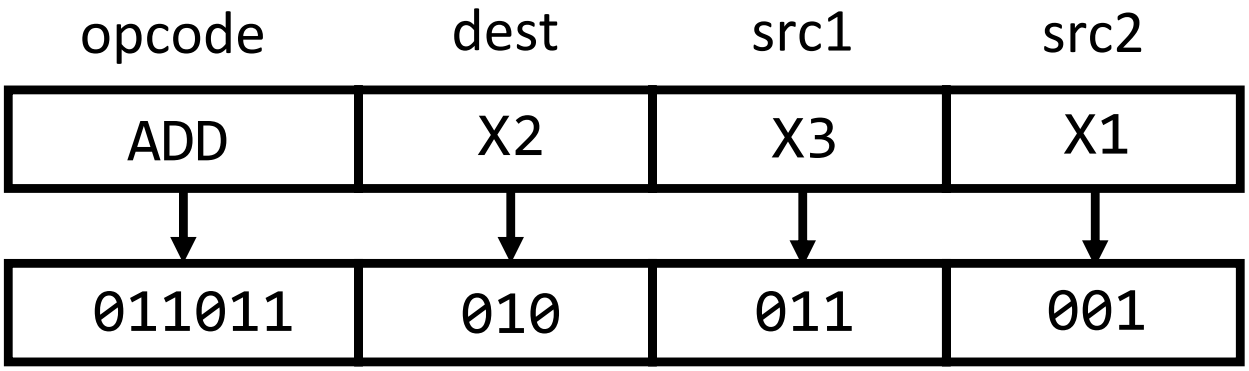
cpu can read and understand.

2^6 = 64 instructions in this architecture.

$01101101001100 1 = 2^0 + 2^3 + 2^4 + 2^7 + 2^9 + 2^{10} + 2^{12} + 2^{13}$ = *13977*

Assembly – Register Addressing

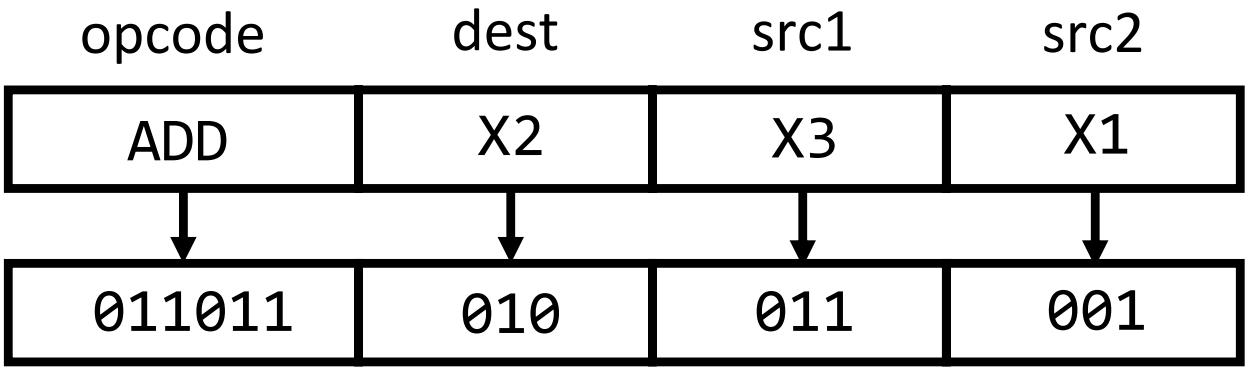
Example ISA (Simplified)



Using 6 bits, how many opcodes can this ISA implement?

Assembly – Register Addressing

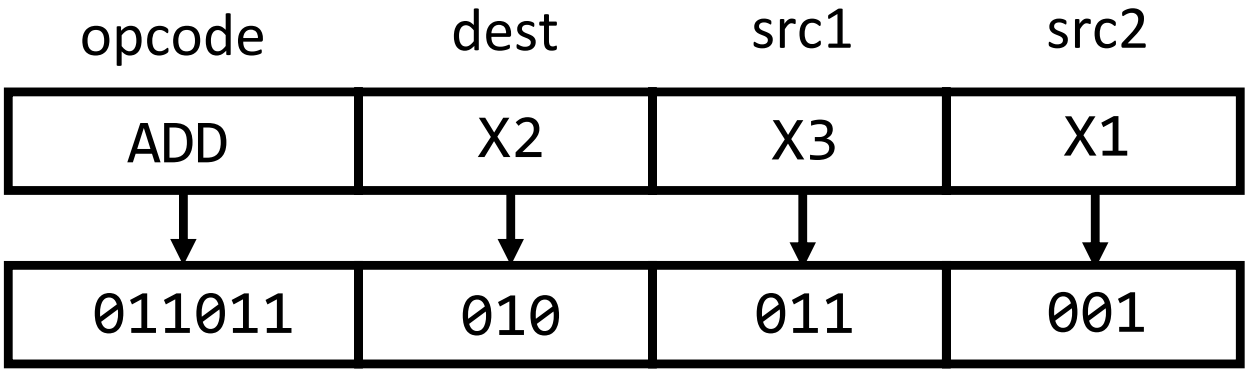
Example ISA (Simplified)



Using 6 bits, how many opcodes can this ISA implement?

Assembly – Register Addressing

Example ISA (Simplified)



Using 6 bits, how many opcodes can this ISA implement?

- m bits can encode 2^m different values
- n values can be encoded in $\lceil \log_2(n) \rceil$ bits
- For above
  - Can encode $2^6 = 64$ opcodes
  - Can encode $2^3 = \mathbf{8}$ src/destination registers

EECS 370 website has a lot of video tutorials, including binary representation https://www.youtube.com/watch?v=KGPfymjE2z8&feature=youtu.be

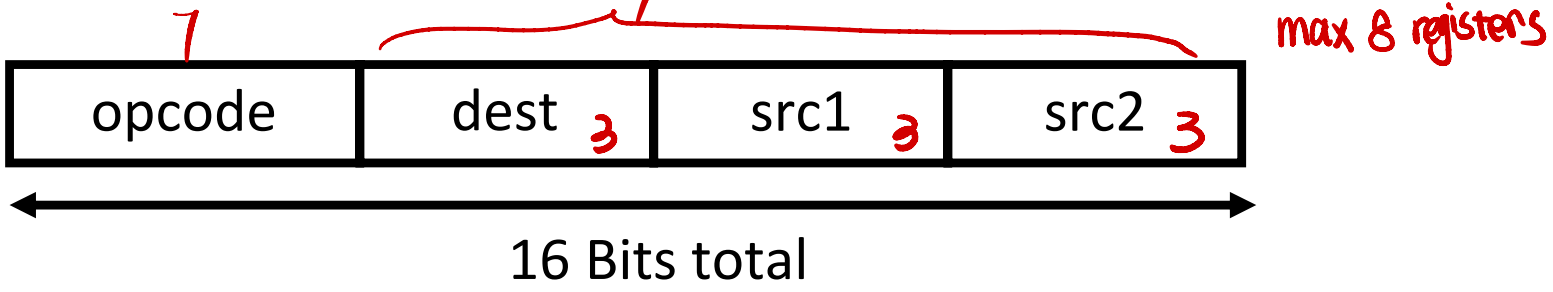
Instruction Encoding – Example 1

Example ISA (Simplified)

What is the max number of registers that can be designed in a machine given:

* 16-bit instructions
* Num. opcodes = 100
* All instructions are (reg, reg) → reg

(i.e., 2 source operands, 1 destination operand, all operands can access all registers)



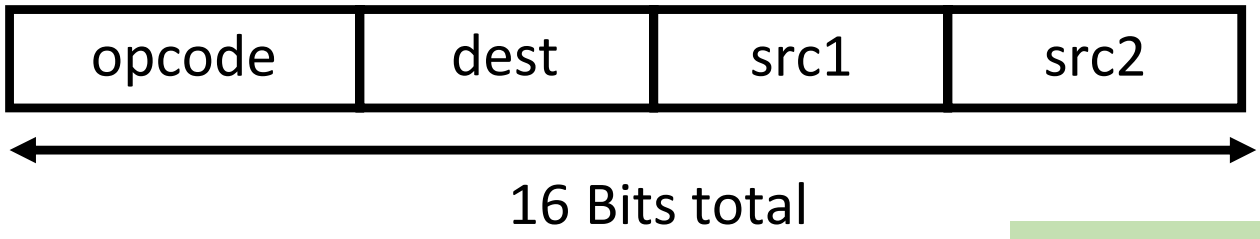
Instruction Encoding – Example 1

Example ISA (Simplified)

What is the max number of registers that can be designed in a machine given:

* 16-bit instructions
* Num. opcodes = 100
* All instructions are (reg, reg) → reg

(i.e., 2 source operands, 1 destination operand, all operands can access all registers)

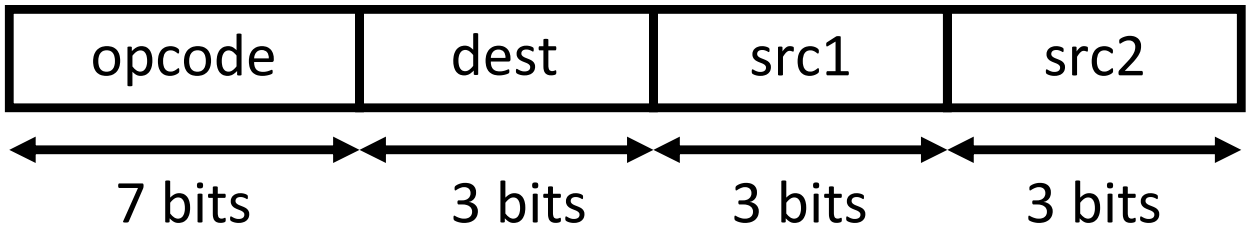


1.num opcode bits = $\lceil \log2(100) \rceil$ = 7
2.num bits for operands = 16 –7 = 9
3.num bits per operand = 9 / 3 = 3
4.maximum number of registers = 2^3 = 8

Instruction Encoding – Example 2

Example ISA (Simplified)

Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

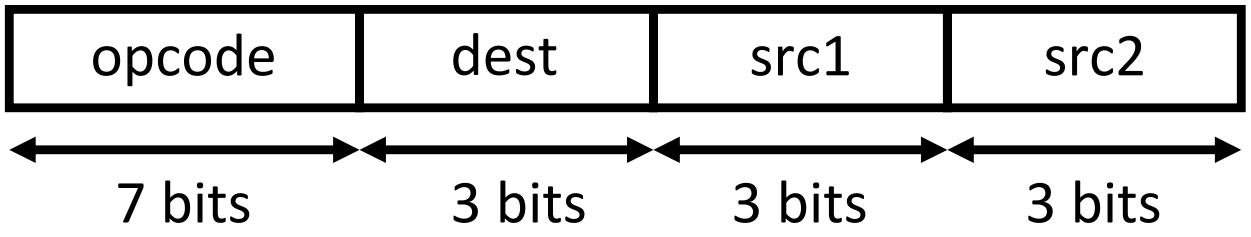
What is the binary / hex / decimal encoding?

	opcode	dest	src1	src2
binary				
hex				
decimal				

Instruction Encoding – Example 2

Example ISA (Simplified)

Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

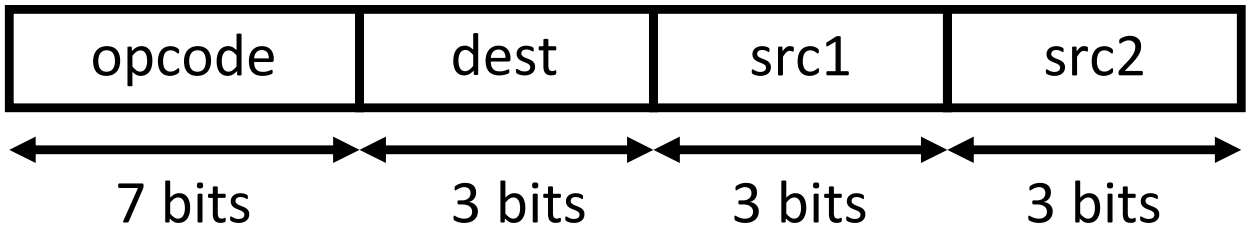
What is the binary / hex / decimal encoding?

	opcode							dest			src1			src2		
binary	0	1	1	0	1	0	1	0	1	0	0	1	1	0	0	1
hex			6			A				9					9	
decimal										27289						

Instruction Encoding – Example 2

Example ISA (Simplified)

Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

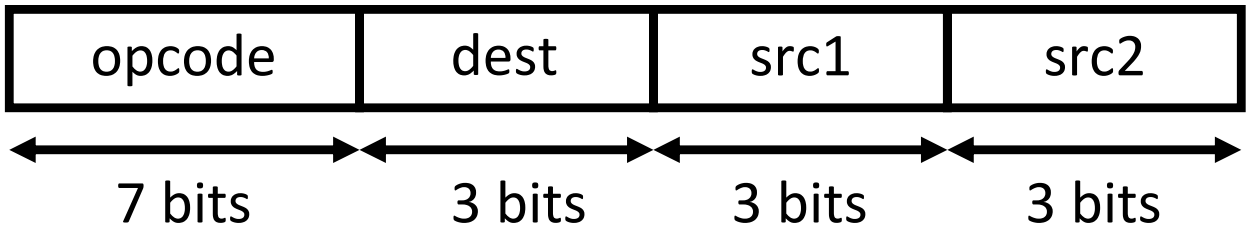
What is the binary / hex / decimal encoding?

	opcode							dest			src1			src2		
binary	0	1	1	0	1	0	1	0	1	0	0	1	1	0	0	1
hex	6				A			9			9					
decimal																

Instruction Encoding – Example 2

Example ISA (Simplified)

Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

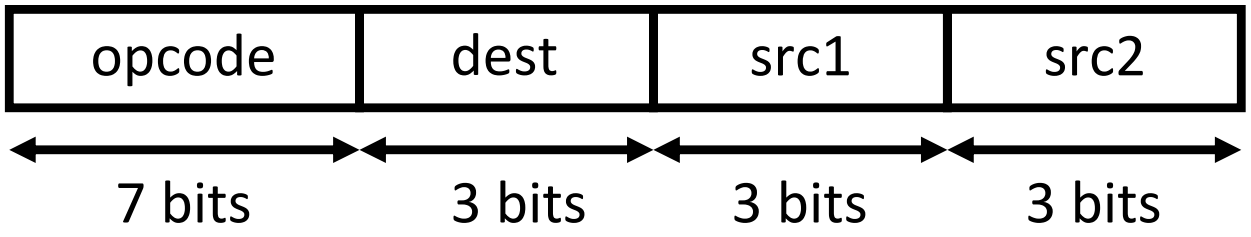
What is the binary / hex / decimal encoding?

	opcode							dest			src1			src2		
binary	0	1	1	0	1	0	1	0	1	0	0	1	1	0	0	1
hex	6				A			9			9					
decimal	27289															

Instruction Encoding – Example 2

Example ISA (Simplified)

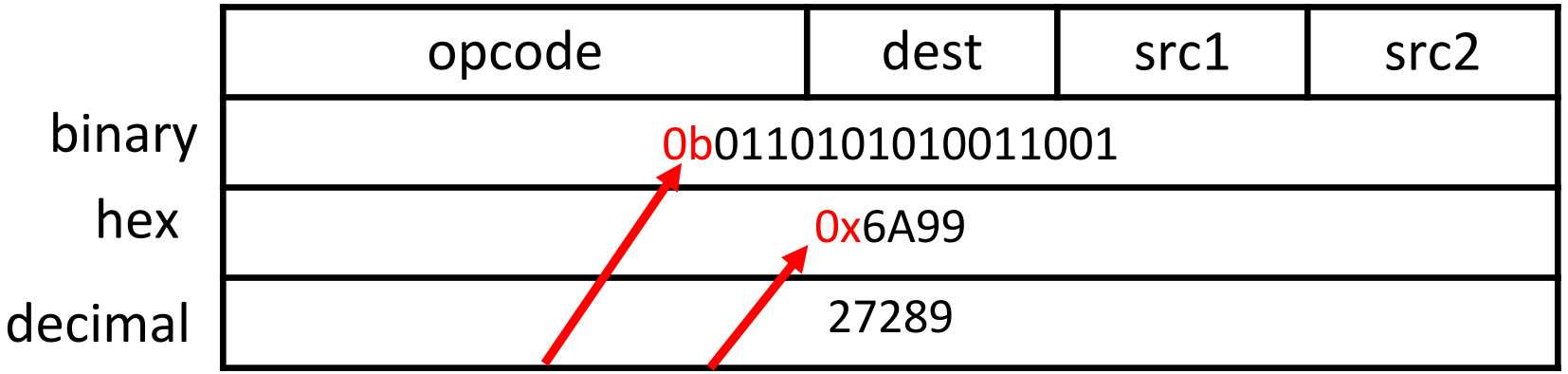
Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

What is the binary / hex / decimal encoding?

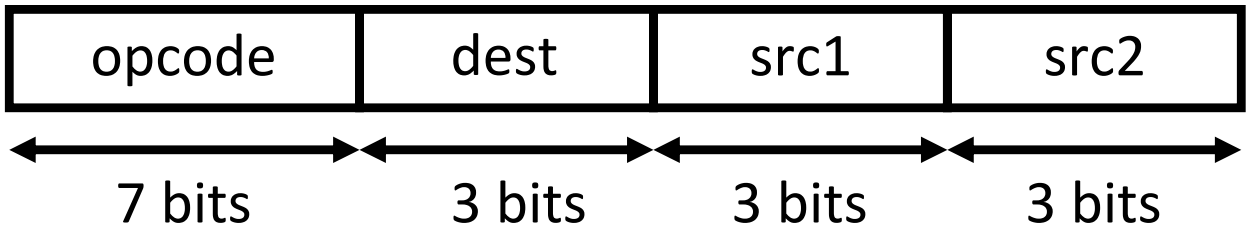


Common prefixes to denote Binary vs Hex vs Decimal

Instruction Encoding – Example 2

Example ISA (Simplified)

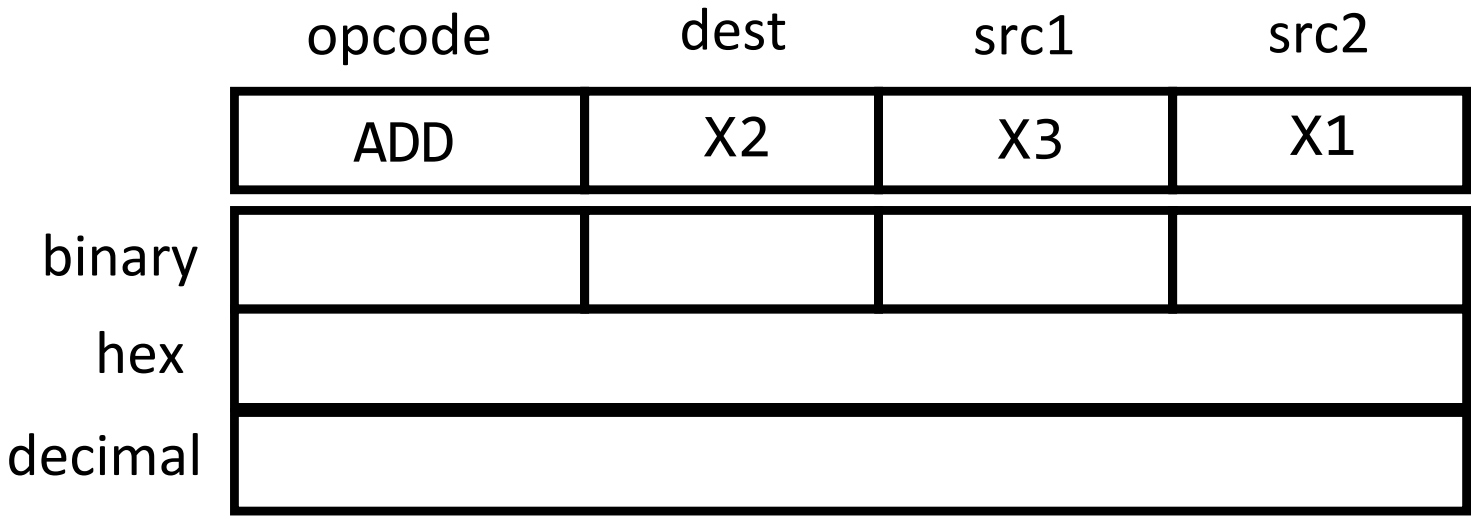
Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

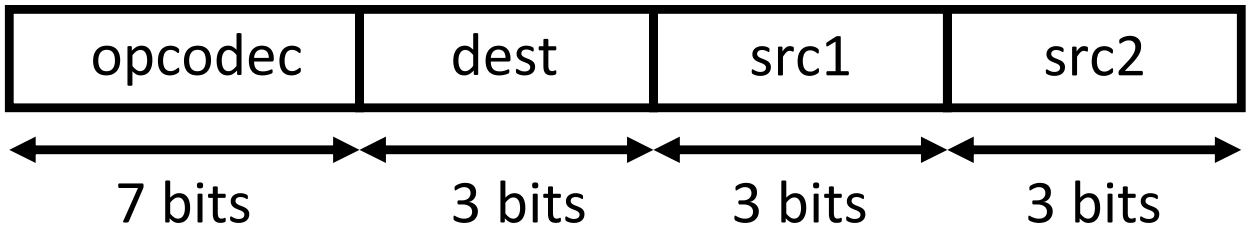
What is the binary / hex / decimal encoding?



Instruction Encoding – Example 2

Example ISA (Simplified)

Given the following ISA instruction fields:



ADD opcode is 53

Register fields encoded with register number

What is the binary / hex / decimal encoding?

	opcode	dest	src1	src2
	ADD	X2	X3	X1
binary	011 0101	010	011	001
hex	0x6A99			
decimal	27289			

L2_3 Assembly Instruction Decoding

converte machine code ⟶ human readable code

EECS 370 – Introduction to Computer Organization – Winter 2022

Learning Objectives

- To understand the process of decoding an assembly instruction
  - Converting from machine to assembly code

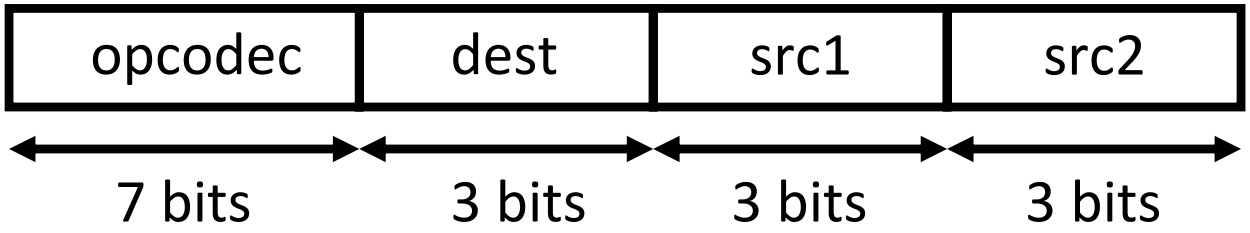
- After completing this video and associated worksheet:
  - You should be able to decode machine code instructions, necessary for Project 1

Instruction Decoding - Example

Example ISA (Simplified)

- Decoding: Given a machine instruction in decimal, convert to assembly

decimal | 27292 |



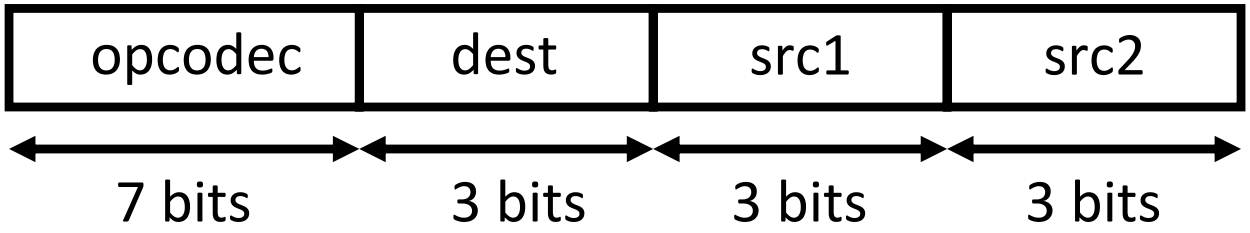
What steps are used to decode a machine code instruction?

Instruction Decoding - Example

Example ISA (Simplified)

- Decoding: Given a machine instruction in decimal, convert to assembly

decimal | 27292 |



1. Convert to binary
2. Separate into fields
3. Convert to decimal
4. Convert assembly instruction fields

1. Convert to Binary

Example ISA (Simplified)

- Given a machine instruction in decimal, convert to binary

decimal | 27292 |

1. Convert to Binary

Example ISA (Simplified)

- Given a machine instruction in decimal, convert to binary

decimal | 27292 |

1. Convert to Binary

Example ISA (Simplified)

- Given a machine instruction in decimal, convert to binary

decimal | 27292 |

Convert with powers of two:

$27292 = 2^{14}(16384) + 2^{13}(8192) + 2^{11}(2048) + 2^{9}(512) + 2^{7}(128) + 2^{4}(16) + 2^{3}(8) + 2^{2}(4) =$ `0110 1010 1001 1100`

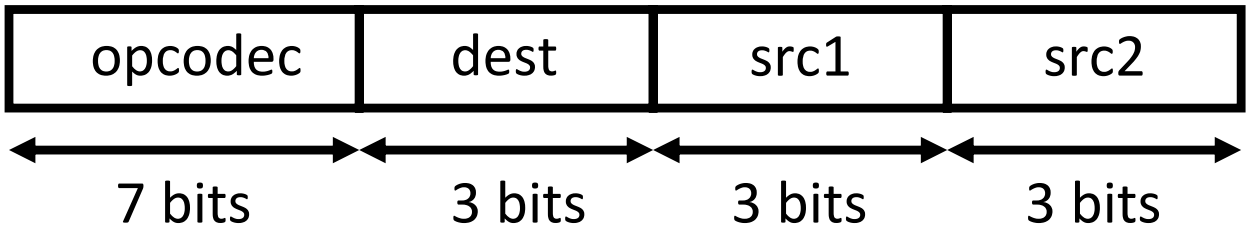
2. Separate into Fields

Example ISA (Simplified)

- Given a machine instruction in binary, separate into fields

decimal | 27292 |

0110 1010 1001 1100



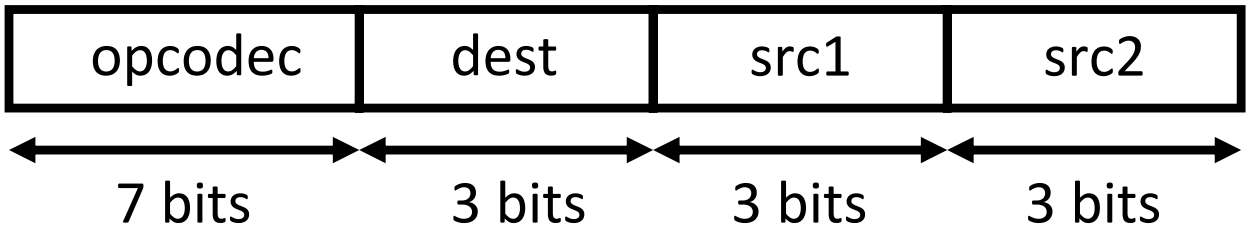
2. Separate into Fields

Example ISA (Simplified)

- Given a machine instruction in binary, separate into fields

decimal | 27292 |

0110 1010 1001 1100



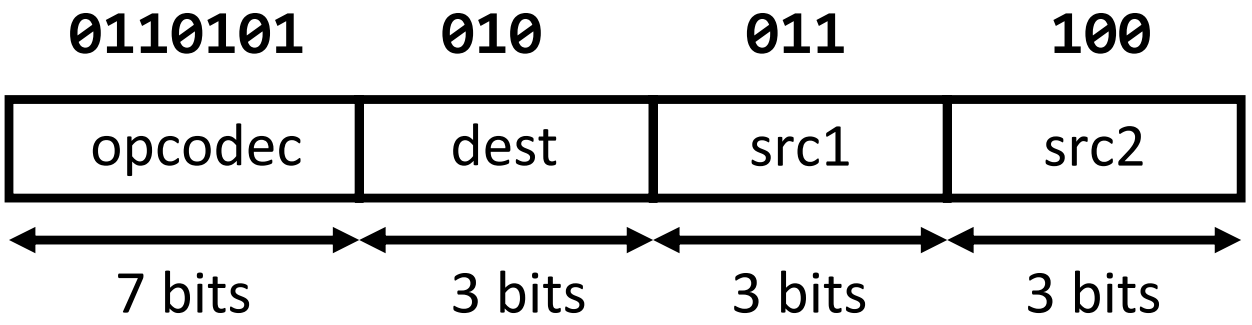
2. Separate into Fields

Example ISA (Simplified)

- Given a machine instruction in binary, separate into fields

decimal | 27292 |

0110 1010 1001 1100



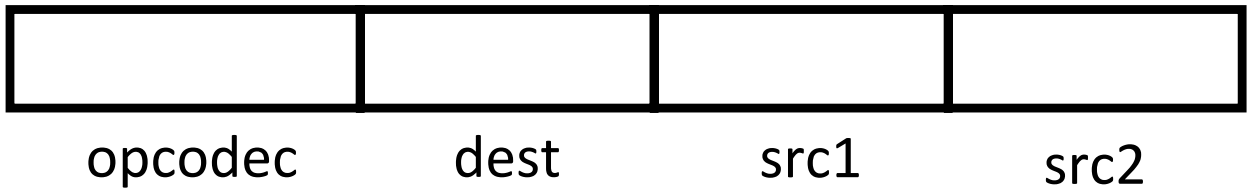
3. Convert Fields to Decimal

Example ISA (Simplified)

- Given a machine instruction in binary in fields, convert to decimal

decimal | 27292 |

0110101 **010** **011** **100**



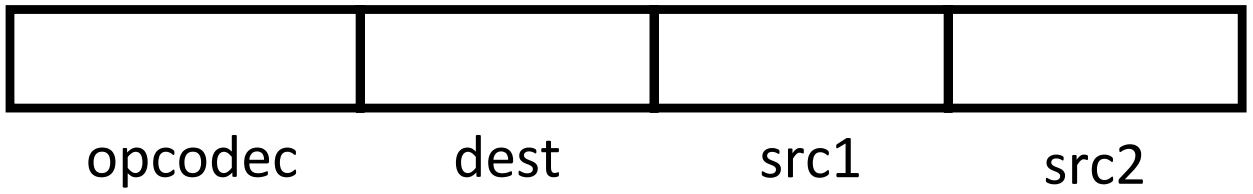
3. Convert Fields to Decimal

Example ISA (Simplified)

- Given a machine instruction in binary in fields, convert to decimal

decimal | 27292 |

0110101 **010** **011** **100**



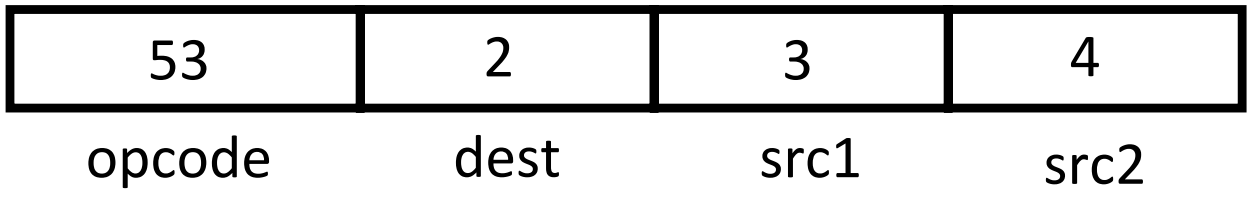
3. Convert Fields to Decimal

Example ISA (Simplified)

- Given a machine instruction in binary in fields, convert to decimal

decimal | 27292 |

0110101 010 011 100



4. Convert to Assembly

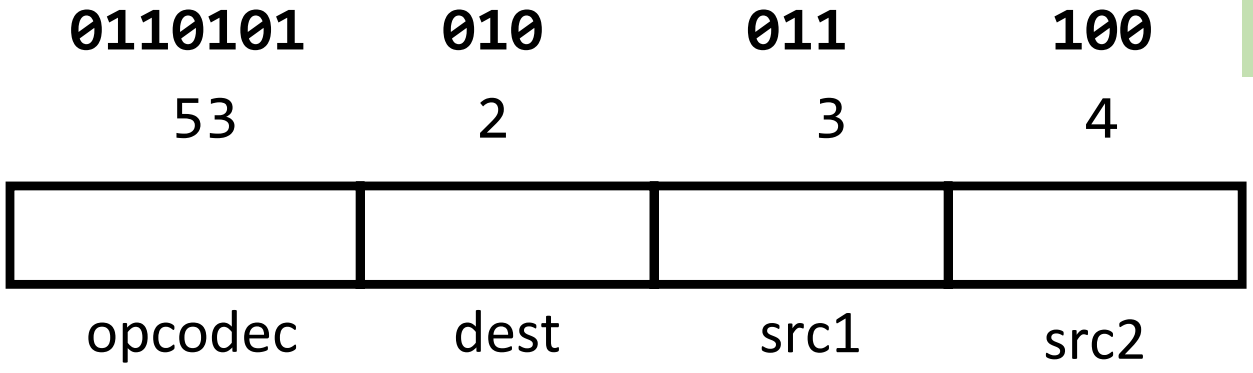
Example ISA (Simplified)

- Given a machine instruction in fields in decimal, convert to assembly

decimal | 27292 |

From previous example:

ADD opcode is 53



4. Convert to Assembly

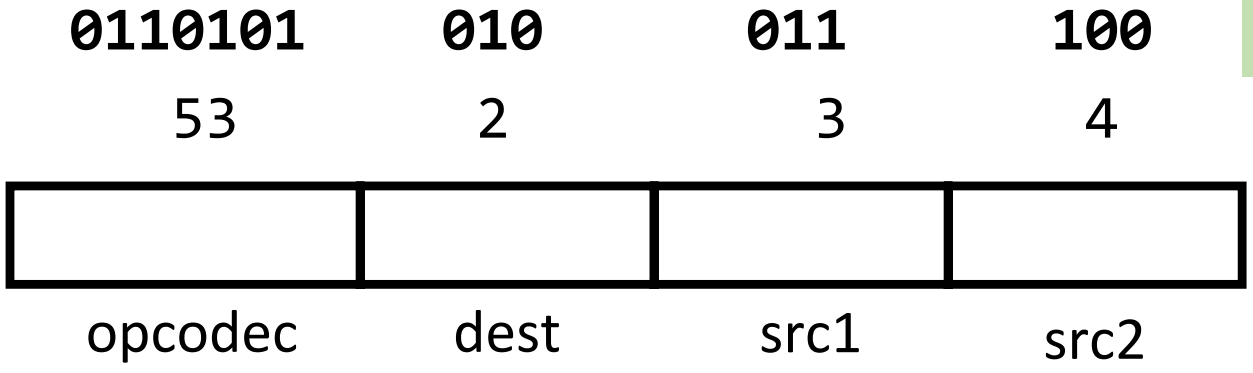
Example ISA (Simplified)

- Given a machine instruction in fields in decimal, convert to assembly

decimal | 27292 |

From previous example:

ADD opcode is 53



4. Convert to Assembly

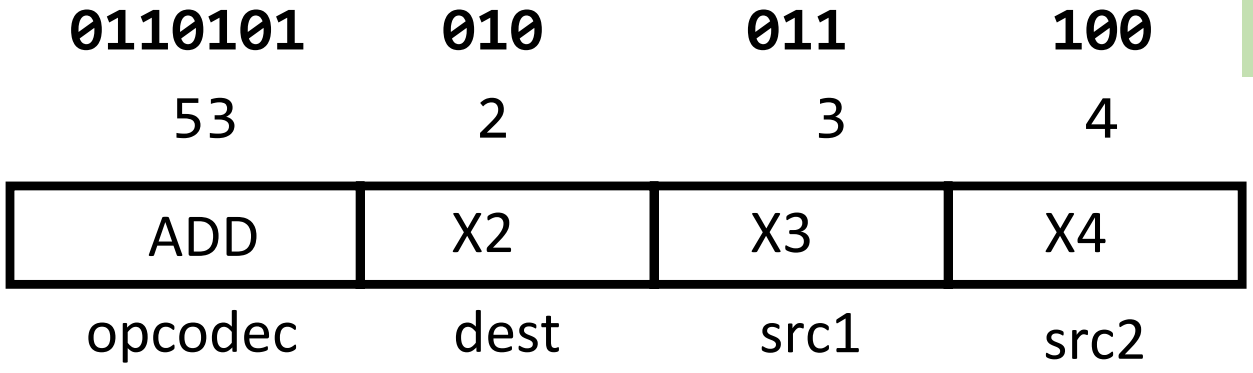
Example ISA (Simplified)

- Given a machine instruction in fields in decimal, convert to assembly

decimal | 27292 |

From previous example:

ADD opcode is 53



Decoding Example 2: LC-2K

LC-2K ISA

Decode LC-2K machine code to LC-2K assembly: 16842754

Decoding Example 2: LC-2K

LC-2K ISA

Decode LC-2K machine code to LC-2K assembly: 16842754

Logistics

- There is one worksheet for lecture 2
  - One exercise on encoding, one for decoding